Constructive Induction on Domain Information

James P. Callan
Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003 U.S.A.

COINS Technical Report 90-14
February 21, 1990

Abstract

One obstacle to wider use of inductive learning algorithms in problem-solving systems is the sensitivity of the algorithms to the way in which examples of the concept are represented. Humans normally decide how the examples will be represented, so success in incorporating inductive learning algorithms varies from person to person. Constructive induction reduces, but does not eliminate, this sensitivity.

An ideal solution would eliminate the need for any human intervention in determining how a problem-solving system and an inductive learning algorithm are integrated. This paper shows how a problem-solver can use its domain knowledge to automatically create a representation of examples that is adequate for learning search control knowledge. The resulting representation describes the examples in terms of how and how well they satisfy the problem-solver's goals. Experimental evidence from two domains is presented to support the claim that this approach is generally useful.

A revised version of this report appeared, under the same title, in *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 614-619) Anaheim, CA: Morgan Kaufmann, July, 1991.

1 Introduction

One of the original goals of Artificial Intelligence (AI) was to endow machines with the ability to learn from experience. Research toward this goal has produced a variety of inductive learning algorithms (e.g. the Perceptron [Minsky & Papert, 1972], AQ11 [Michalski & Chilausky, 1980], ID3 [Quinlan, 1983]) and some impressive performance programs (e.g. Samuel's checker program [Samuel, 1959], MetaDendral [Buchanan & Mitchell, 1978], LEX [Mitchell, Utgoff & Banerji, 1983]). However, in spite of these successes, few of today's problem-solving systems have the ability to learn.

One possible reason that inductive learning algorithms are not more widely used in problem-solving systems is their sensitivity to the way in which information is represented. Some of the best-known successes in machine learning have been due in part to careful or fortunate choices of representations (e.g. AM [Lenat & Brown, 1984]). When the same algorithms are applied with less-carefully chosen vocabularies, they may fail to learn anything useful. As a result, the success in using inductive learning algorithms varies from person to person.

Constructing a good vocabulary 'by hand' can take a long time, because it is essentially a manual search of the space of possible vocabularies. One well-known example is the two man-months it took Quinlan to design a vocabulary that enabled ID3 to learn certain chess positions [Quinlan, 1983]. Constructive induction reduces the sensitivity of an inductive algorithm to its vocabulary by enabling the algorithm to dynamically construct new terms [Michalski, 1983]. Constructive induction is a much faster way to search the space of possible vocabularies. However, the initial vocabulary is still important, because it partially determines the size and topology of the space of possible vocabularies.

Creation of the initial vocabulary has remained a manual process, partly because creating such vocabularies requires knowledge that is not normally available to an inductive learning algorithm. This paper presents an alternate solution. It shows how a problem-solver can use what it knows about a search problem to generate an initial vocabulary for describing search states to an inductive learning algorithm. The resulting vocabulary describes search states in terms of how and how well they satisfy the problem-solver's goals and subgoals.

The following section describes the importance of the initial vocabulary in more detail. Section 3 describes the algorithm for creating an initial vocabulary from domain knowledge. Section 4 provides performance results for the vocabularies that are automatically generated for two different problems. Section 5 discusses some unanswered questions about this method of generating vocabularies, and how they might be answered. Finally, Section 6 discusses the significance of the work and concludes.

2 The Initial Vocabulary

The goal of the research presented here is the automatic construction of a good initial vocabulary for inductive learning. The source of the initial vocabulary is assumed to be the problem-solving system that supplies the examples of the target concept. Therefore, the vocabulary must be based upon information available to the problem-solver. The most easily available information is the vocabulary that the problem-solver uses in its own descriptions

of objects in the domain. Unfortunately, a vocabulary that is designed for a particular problem-solver may be quite inappropriate for a learning algorithm [Flann & Dietterich, 1986].

The simplest approach to fix an inadequate vocabulary is to add every new term that might possibly be useful. This approach is effective in domains where the number of possible terms is small. However, in some domains the number of possible terms is quite large or infinite. The dimensionality of the space searched by an inductive learning algorithm (the *concept space*) is determined by the size of the vocabulary, so large vocabularies result in large search spaces. In general, large vocabularies cause slower learning; in some cases, the concept that is learned is also inferior to a concept that is learned with just a subset of the vocabulary [Saxena, 1989]. Therefore, it is preferable to add only the terms that facilitate the discovery of a useful concept.

Most systems that perform constructive induction conduct a separate heuristic search through a space of possible new terms (the *new term space*), keeping those that seem most effective, and discarding the rest (e.g. STAGGER [Schlimmer & Granger, 1986], CITRE [Matheus & Rendell, 1989]). In the new term space, each state represents a new term that can be created. The initial vocabulary determines both the starting state and the dimensionality of the new term space. A good initial vocabulary produces a small new term space in which the initial state is not too far from states representing useful terms. A poor initial vocabulary prevents even the best of the available constructive induction algorithms from finding useful new terms.

3 Use of Domain Information

There are many different problem-solvers, with many different architectures; what they know, and in what detail they know it, varies. However, one very general and accepted view is that all problem-solvers conduct a search through some problem space [Newell & Simon, 1972]. While different problem-solvers may conduct the search quite differently, this paradigm provides a convenient framework for describing the problem-solver's knowledge and intentions. At a minimum, the problem-solver can be assumed to have declarative knowledge about its goal(s) and the operators available for transforming search states, and it can be assumed to be trying to reach a goal state along some minimal cost path.

These assumptions about the problem-solver's knowledge and intentions serve two purposes. The assumption about its knowledge defines a source of domain information. However, the objects in most domains can be described in a variety of ways. No single description is 'best' for every purpose. Therefore the assumption about the problem-solver's intent is necessary, because it defines the purpose. A vocabulary for describing objects will be considered 'good' if it makes it easier for the learning algorithm to recognize states that lead to a goal state.

What types of terms might help a learning algorithm recognize states that lead to a goal state? This problem is ill-formed in the general case. However, certain types of terms are often useful, and arise repeatedly in both human problem-solving and the AI literature. Two are of particular interest because they can be created automatically from the information available from the problem-solver. They are the *degree of satisfaction* and the *method of*

satisfaction. Each is discussed in more detail below.

3.1 Degree of Satisfaction

The *degree of satisfaction* indicates how similar a given state is to a goal state. A single term measuring degree of satisfaction is useful for hillclimbing if it provides a gradient that can be followed. A set of such terms can be created by measuring how similar the state is to states that achieve subgoals. Such terms are useful in many problem-solving strategies, but means-ends analysis [Newell & Simon, 1972] and relaxation specifically require them. The importance of information about degree of satisfaction in heuristics for controlling search supports the theory that it will be useful for learning search control knowledge.

The simplest measure of degree of satisfaction is a type of term called *DS0*. A *DS0* term is boolean. Its value in a given state s is T if s satisfies a particular subgoal, and F if it does not. The major limitation of *DS0* terms is that their granularity is coarse. The value of a single term in a given state is often identical to its value for the parent state.

The cost of evaluating a *DS0* term depends upon the form of the subgoal from which it is derived. If the subgoal contains no quantifiers, as in $p(l_1, \ldots, l_n)$, the cost is one evaluation of the predicate p. If the subgoal contains quantifiers, as in $\forall v_1 \epsilon S_1, \ldots, \forall v_n \epsilon S_n, p(v_1, \ldots, v_n)$, the average cost is $(\Pi_{i=1}^{n} |S_i|)/2$ evaluations of the predicate p.

A type of term called *DS1* is a more detailed measure of degree of satisfaction. *DS1* terms are real-valued, and can be created for any subgoal that begins with a universal quantifier. Subgoals with universal quantifiers require that a set of objects satisfy some condition. For a subgoal with a single universal quantifier, the degree of satisfaction is the percentage of objects in the set that satisfy the condition. More generally, for subgoals that have multiple universal quantifiers, like $\forall v_1 \epsilon S_1, \ldots, \forall v_n \epsilon S_n, p(v_1, \ldots, v_n)$, the degree of satisfaction is the percentage of permutations of objects that satisfy the predicate p.

For example, if a subgoal for a blocks-world problem requires that all blocks be on a table ($\forall b \epsilon B, \mathrm{on}(b, \mathrm{Table})$), then the corresponding *DS1* term indicates the percentage of blocks on the table.

DS1 terms are a subset of the terms produced by Michalski's *counting arguments* rules [Michalski, 1983]. The counting arguments rules are applied to concept descriptions that contain quantified variables. Each counting arguments rule counts the permutations of objects that satisfy some condition. However, the source of the conditions is not specified, suggesting that the source is a human. In the method proposed here, subgoals act as the source of the conditions, so human intervention is not necessary.

3.2 Method of Satisfaction

The *method of satisfaction* indicates *how* a given state satisfies a subgoal. If the state only partially satisfies the subgoal, then attention is restricted to the parts that satisfy the subgoal. Terms that measure the method of satisfaction are necessary for heuristics that give preference to choices that were successful in the past. One way of implementing such a heuristic is to favor states in which the degree of satisfaction increases but the method of satisfaction remains constant.

The simplest measure of method of satisfaction is a type of term called *MS0*. *MS0* terms can be created for subgoals that require every permutation of objects to satisfy some constraint; these correspond to subgoals that begin with a universal quantifier. The *MS0* term for a subgoal specifies the average, or prototypical, way in which the constraint is satisfied. If the constraint is an inequality, then the corresponding *MS0* term specifies the average size of the inequality.

For example, if a subgoal for the Eight Queens problem (discussed in section 4.1) requires that no two queens occupy the same column ($\forall q_1, q_2 \epsilon Q, \text{column}(q_1) \neq \text{column}(q_2)$), then the corresponding *MS0* term indicates the average distance, in columns, between each pair of queens.

3.3 Finding Subgoals

Some problem-solvers have detailed information about the problems that they try to solve. For example, problem-solvers with blackboard architectures are designed to decompose the description of a goal state into subgoals. These problem-solvers could simply provide a list of subgoals as part of the domain information. Such a list might be more useful or more relevant than anything that a knowledge-free decomposition procedure would yield.

Other problem-solvers, for example those with simpler search algorithms, may have little or no explicit information about subgoals. Therefore, some domain-independent method must be available for decomposing the specification of a goal into a set of subgoal specifications.

One very simple approach is to repeatedly split the specification of the goal state along conjuncts and disjuncts, following the usual precedence rules. For example, the expression $a \wedge (b \vee c) \wedge d$ yields subgoals a, $b \vee c$, and d in the first iteration; it yields subgoals b and c in the second iteration. In this example, five subgoals are created. In general, this simple heuristic produces between $c+1$ and $2c$ subgoals, where c is the number of connectives ($\wedge, \vee$) in the original expression.

This simple approach, while new, produces expressions that Gaschnig called *edge subgraphs* [Gaschnig, 1979] and Pearl called *relaxed models* [Pearl, 1984]. Each subgoal is a simpler, or less constrained, problem whose solution can be used as a heuristic in solving the original problem. This can be characterized as a *lookahead* approach, because it requires searching ahead for a solution to a subgoal. This additional search effort usually makes the cost of the heuristic more expensive than blind search [Valtorta, 1983], although recent work shows that it can sometimes be made acceptable [Mostow & Prieditis, 1989]. This cost problem, if it is a problem, does not affect the methods reported in this paper, because they are not based upon searching for solutions to subgoals.

3.4 Use of Search Space Operators

The previous sections have assumed that all of the information about the goal state can be found in its description. However, that may not be the case in practice. It is usually desirable to move as many of the goal state requirements as possible into the preconditions of the search space operators, so that fewer search states are generated. The advantage of doing

so is that the search speed may increase dramatically; the disadvantage is that information about the goal state becomes dispersed throughout the problem specification.

This problem disappears if the procedure for generating subgoals is also applied to the preconditions of each search space operator. The resulting expressions may or may not correspond to what a human observer would call a subgoal. However, treating them as subgoals does no harm; at worst, it allows the creation of a few more useless terms. This additional cost is offset by the guarantee that *all* of the information about the goal state will be found, whether it resides in the description of the goal state or in the preconditions of the operators.

4 Results

The algorithm presented in the preceding section has been implemented in a computer program called CINDI. The input to CINDI is a problem specification, expressed in first order predicate calculus. The output from the program is a set of specifications for the new terms.

Testing the quality of the resulting terms is difficult, particularly because there is no commonly accepted method. The approach adopted in this study was to measure how many of the different examples of a concept needed to be seen by a single linear threshold unit (LTU) [Nilsson, 1965] for it to reach a particular level of accuracy in identifying which of two randomly selected search states is closer to a goal state. The LTU was chosen because it handles real-valued terms and because of its well-known limitations [Minsky & Papert, 1972]. The intent was to study the effect of the *representation*, rather than the power of a particular learning algorithm. More complex learning algorithms use multiple hyperplanes to classify examples, while an LTU uses just one. Therefore the vocabularies that an LTU prefers would also be useful to more complex learning algorithms.

LTUs are often associated with linearly separable examples, but linear separability is not strictly necessary. When the examples are linearly separable, the learning rule is guaranteed to find a hyperplane that separates the positive examples from the negative examples, yielding 100% classification accuracy. When the examples are *not* linearly separable, the least-mean squares learning rule tends to find hyperplanes that minimize the mean-squared error [Dietterich, London, Clarkson & Dromey, 1982]. These hyperplanes provide good, but not perfect, classification.

In each experiment, training and testing were alternated until the LTU reached a specified level of accuracy in classifying examples. Each training phase was performed on only a very small set of randomly chosen examples, but testing was performed on every possible example. This approach provides perfect measurements of how accurate a LTU is in classifying examples, but it is time-consuming. As a result, only two problems have been examined in any detail; they are discussed below. Each measurement reported in the following sections is based upon an average of 100 experiments.[1]

[1]The number 100 was chosen to give confidence in the results. It is not meant to imply large variance in the results.

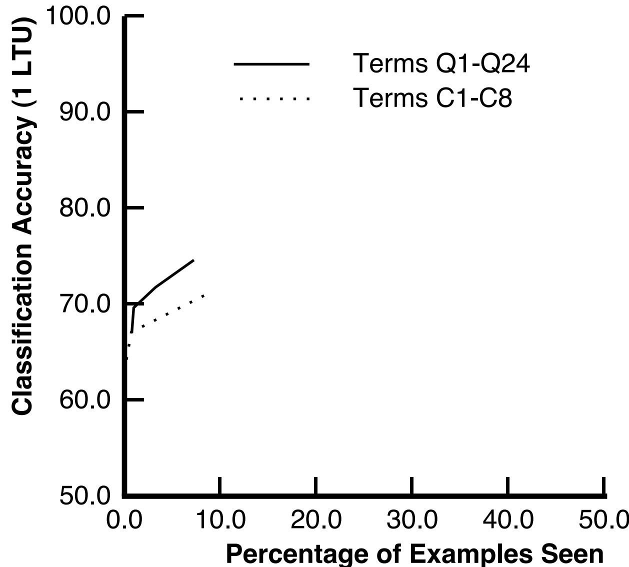

Figure 1: Learning curves for the 8 queens problem.

4.1 The Eight Queens problem

The Eight Queens problem is a constraint satisfaction problem that is commonly used to study the effectiveness of heuristics for search control [Nilsson, 1980; Pearl, 1984]. It requires a problem-solver to find a placement of eight queens on a chess board that prevents any queen from attacking another queen. The problem is difficult because the placement of any single queen affects the placement of every other queen. The relaxation and divide-and-conquer heuristics that are often used in constraint satisfaction problems are ineffective on the Eight Queens problem.

CINDI generated nine subgoals from the specification of the goal state. It generated another subgoal from the specification of the *assign-column* operator.[2] Two of the ten subgoals were syntactically equivalent; one of them was discarded, leaving nine subgoals. All nine subgoals satisfied the requirements enabling *DS0* and *DS1* terms to be created, but only six satisfied the requirements for *MS0* terms. The resulting terms were arbitrarily labeled Q1-Q24.

For example, one of the subgoals created from the specification of the goal state required that no two queens occupy the same column. Three terms were created from that subgoal. Term Q6 was a *DS0* term that indicated whether or not the subgoal was satisfied. Term Q15 was a *DS1* term that indicated the percentage of pairs of placed queens occupying different columns. Term Q21 was an *MS0* term that indicated the average distance, in columns, between each pair of placed queens.

The performance of a single LTU using terms Q1-Q24 was compared with its performance using a hand-coded representation, labeled C1-C8. The hand-coded representation was intended to be the kind of representation that a problem-solver might use internally. Terms C1-C8 were numeric terms that indicated the columns to which each queen was assigned. C1 indicated the column for queen 1, C2 indicated the column for queen 2, and so on. No

[2]See [Callan, 1989] for the specification.

attempt was made to improve this representation, and no other representations were tried.

The learning curve in Figure 1 shows how many of the different examples the LTU had to be trained on for it to reach a given level of accuracy in identifying which of two randomly selected search states is closer to a goal state. The set of terms generated automatically by CINDI clearly outperformed the hand-coded representation when the training set consisted of less than 10% of the different examples. Preliminary results suggest that a single LTU can only achieve about 75% accuracy with either of these representation, no matter how many examples are seen.

4.2 A Blocks-World problem

The blocks-world is a simple domain that is often studied in Artificial Intelligence. In the problem studied here, it contained four blocks, labeled A through D, and a table. The problem-solver's goal was to stack A on B, B on C, C on D, and D on the table. The starting state was assumed to be generated randomly.

CINDI generated four subgoals from the specification of the goal state. It generated another six subgoals from the specification of the *stack-block* and *move-block-to-table* operators. Three of the ten subgoals were syntactically equivalent; two of them were discarded, leaving eight subgoals. All eight subgoals satisfied the requirements enabling *DS0* terms to be created, four satisfied the requirements for *DS1* terms, and none satisfied the requirements for *MS0* terms. The resulting terms were arbitrarily labeled T1-T12.

For example, one of the subgoals created from the *stack-block* operator required that there be a block with an empty top. Two terms were created from that subgoal. Term T6 was a *DS0* term that indicated whether or not the subgoal was satisfied. Term T10 was a *DS1* term that indicated the percentage of blocks that had empty tops.

The performance of a single LTU using terms T1-T12 was compared with its performance using two hand-coded representations, labeled B1-B16 and L1-L4. The hand-coded representations were intended to be the kind of representations that a problem-solver might use internally. Terms B1-B16 were boolean terms that indicated the positions of the blocks. B1 indicated whether A was on B, B2 indicated whether A was on C, and so on. Terms L1-L4 were numeric encodings of the positions of the blocks. The blocks were numbered 0 (for A) through 3 (for D), with 5 representing the table. Term L1 indicated block A's position, term L2 indicated block B's position, and so on. No attempt was made to improve any of the representations, and no other representations were tried.

The learning curve in Figure 2 shows how many of the different examples the LTU had to be trained on for it to reach a given level of accuracy in identifying which of two randomly selected search states is closer to a goal state. The set of terms generated automatically by CINDI clearly outperformed both of the hand-coded representations. A combination of two representations performed best of all.

5 Open Problems

The above results demonstrate that the CINDI algorithm is capable of automatically generating useful initial vocabularies. However, three improvements are desirable. The CINDI

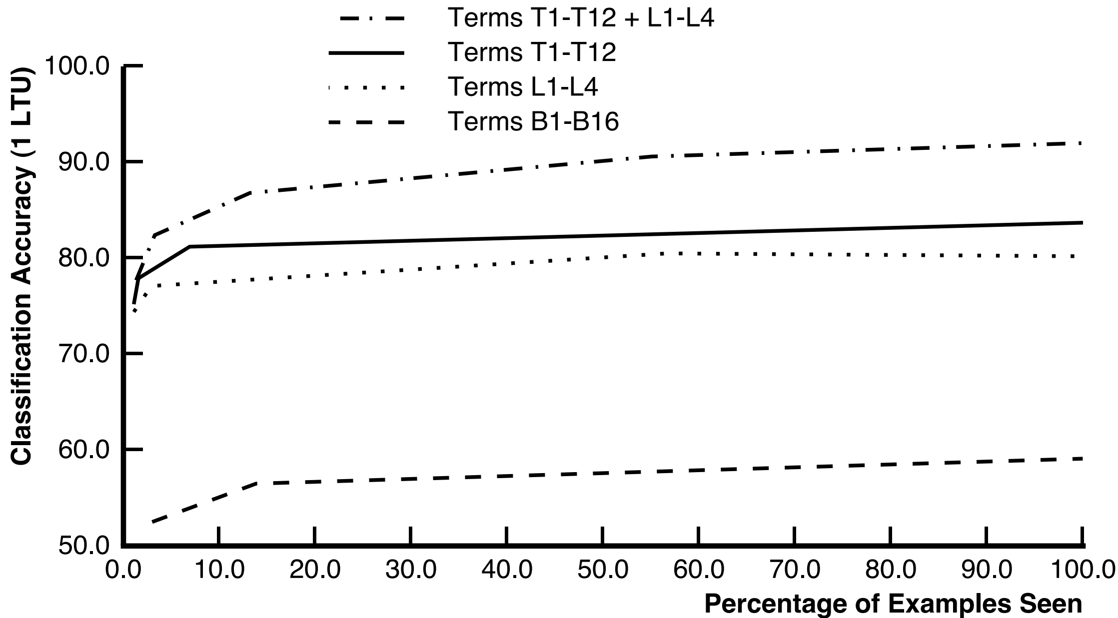

Figure 2: Learning curves for the 4 blocks problem.

algorithm would be more effective if it had more methods of generating terms, if the resulting terms could completely distinguish between positive and negative examples, and if the weak or useless terms could be discarded.

More methods of generating new terms from domain information are necessary for two reasons. First, the set of terms generated by the existing methods is not always capable of completely discriminating between the positive and negative examples of a concept. This inability is one reason that the LTU's did not achieve 100% accuracy on the problems in Section 4. Second, some of the methods in Section 3 are only applicable to certain types of subgoals. A wider range of methods would make CINDI less sensitive to the way in which domain information is represented.

One way to expand the types of new terms is to generalize the *MS0* method to include a wider range of constraints. For example, it may be possible to create *MS0* terms for subgoals with symbolic constraints, by using the mode instead of the average.[3]

Another way of enabling a vocabulary to completely discriminate between positive and negative examples is to provide it with all of the terms used by the problem-solver. These are readily available in the domain information, and could be extracted easily. Adding such terms would guarantee the ability to discriminate between positive and negative examples. It might also improve learning performance, as it did in the combined representation (T1-T12 + L1-L4) of Figure 2. The disadvantage, of course, is that it would also increase the size of the space being searched by the learning algorithm.

Adding additional terms would be less of a problem if it were possible to discriminate between the necessary and unnecessary terms. Unfortunately, the best algorithms for evaluating new terms have requirements that cannot be satisfied without placing restrictions on how the problem-solver performs its search. Saxena's method requires that terms have discrete values [Saxena, 1989], while Mehra's method requires a very small set of representative

[3]The mode is the value, in a list of values, that occurs most often.

examples [Mehra, Rendell & Wah, 1989]. Therefore, the current strategy is to generate the terms described in Section 3 and let the inductive learning algorithm decide which ones to use. This strategy works because the CINDI algorithm does not create many new terms. However, it will become less effective as other methods of generating terms are added, so an alternative must be found.

None of these problems prevents the use of the CINDI algorithm. However, solutions to any of them would improve its effectiveness.

6 Conclusion

Machine learning algorithms will be more broadly accessible when they can be incorporated into problem-solving systems without human intervention. The empirical results reported here demonstrate that, for the two problems investigated, a good initial vocabulary can be created automatically from information available to the problem-solver. The experiments also show that the resulting initial vocabularies are better for learning than vocabularies that the problem-solver might use in its own work.

One might argue that this approach to generating an initial vocabulary does not solve the representation problem; instead, it merely moves it back into the problem-solver. However, initial vocabularies cannot be made out of nothing. Every constructive induction algorithm is sensitive to how its sources of information are represented. This sensitivity is inevitable, and perhaps desirable. The learning algorithm is, after all, supposed to aid the problem-solver in its task. Therefore it may not be unreasonable that it is influenced by the way in which the problem-solver views its task.

Rather than avoiding the representation problem, this approach directly addresses a part of it that has traditionally been avoided in the inductive learning paradigm. The initial vocabulary is crucial to the success of inductive learning, but its source is always left unspecified. The CINDI algorithm fills this void. It shows that, with a few assumptions about the problem-solving environment, domain information can be used to generate the initial vocabulary for an inductive learning algorithm.

Acknowledgements

This research was supported by a grant from Digital Equipment Corporation, and by the Office of Naval Research through a University Research Initiative Program, under contract N00014-86-K-0764. I thank Paul Utgoff, Sharad Saxena, Tom Fawcett, David Haines, Carla Brodley, Margie Connell, David Lewis and Rick Yee for their comments on drafts of this report.

Bibliography

Buchanan, B. G., & Mitchell, T. M. (1978). Model-directed learning of production rules. In Waterman & Hayes-Roth (Eds.), *Pattern-directed inference systems*. New York: Academic Press.

Callan, J. P. (1989). Knowledge-based feature generation. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 441-443). Ithaca, NY: Morgan Kaufmann.

Dietterich, T. G., London, B., Clarkson, K., & Dromey, G. (1982). Learning and inductive inference. In Cohen & Feigenbaum (Eds.), *The Handbook of Artificial Intelligence: Volume III.* San Mateo, CA: Morgan Kaufmann.

Flann, N. S., & Dietterich, T. G. (1986). Selecting appropriate representations for learning from examples. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 460-466). Philadelphia, PA: Morgan Kaufmann.

Gaschnig, J. (1979). A problem similarity approach to devising heuristics: First results. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* (pp. 301-307). Tokyo, Japan.

Lenat, D. B., & Brown, J. S. (1984). Why AM and EURISKO appear to work. *Artificial Intelligence, 23*, 269-294.

Matheus, C. J., & Rendell, L. A. (1989). Constructive induction on decision trees. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 645-650). Detroit, Michigan: Morgan Kaufmann.

Mehra, P., Rendell, L. A., & Wah, B. W. (1989). Principled constructive induction. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 651-656). Detroit, Michigan: Morgan Kaufmann.

Michalski, R. S., & Chilausky, R. L. (1980). Learning by being told and learning from examples: An experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *Policy Analysis and Information Systems, 4*, 125-160.

Michalski, R. S. (1983). A theory and methodology of inductive learning. In Michalski, Carbonell & Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* San Mateo, CA: Morgan Kaufmann.

Minsky, M., & Papert, S. (1972). *Perceptrons: An introduction to computational geometry (expanded edition).* Cambridge, MA: MIT Press.

Mitchell, T. M., Utgoff, P. E., & Banerji, R. B. (1983). Learning by experimentation: Acquiring and refining problem-solving heuristics. In Michalski, Carbonell & Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* San Mateo, CA: Morgan Kaufmann.

Mostow, J., & Prieditis, A. E. (1989). Discovering admissible heuristics by abstracting and optimizing: A transformational approach. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 701-707). Detroit, Michigan: Morgan Kaufmann.

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Nilsson, N. J. (1965). *Learning machines.* New York: McGraw-Hill.

Nilsson, N. J. (1980). *Principles of artificial intelligence.* Palo Alto, CA: Tioga.

Pearl, J. (1984). *Heuristics: Intelligent search strategies for computer problem solving.* Reading, Ma: Addison-Wesley.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In Michalski, Carbonell & Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* San Mateo, CA: Morgan Kaufmann.

Samuel, A. (1959). Some studies in machine learning using the game of Checkers. *IBM Journal of Research and Development, 3*, 211-229.

Saxena, S. (1989). *The input representation problem in machine learning*, (unpublished thesis proposal), Amherst, MA: University of Massachusetts, Computer and Information Science Department.

Schlimmer, J. C., & Granger, R. H., Jr. (1986). Incremental learning from noisy data. *Machine Learning, 1*, 317-354.

Valtorta, M. (1983). A result on the computational complexity of heuristic estimates for the A* algorithm. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (pp. 777-779). Karlsruhe, West Germany: William Kaufmann.